Portable and Scalable MPI Shared File Pointers using IOFSL

P. Beckman, J. Bent, J. Cope, G. Grider, K. Iskra, T. Jones,
D. Kimpe, S. Poole, J. Nunez, R. Ross, L. Ward

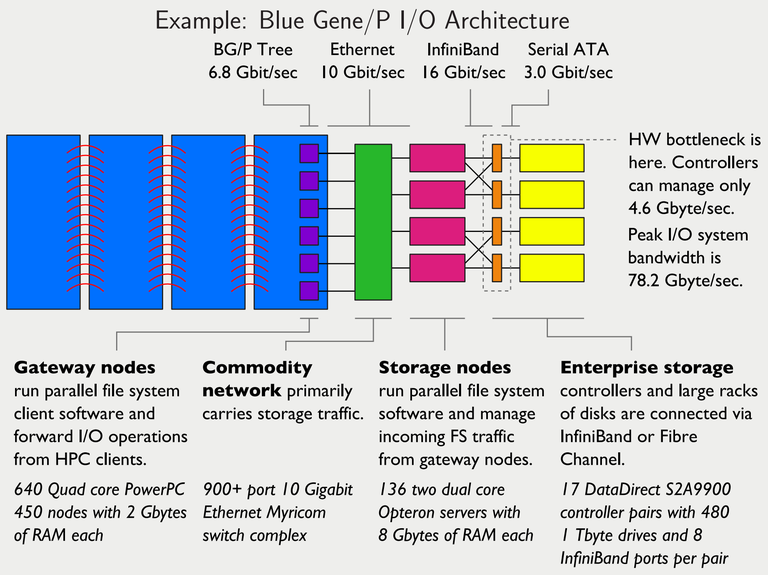
Argonne National Laboratory, Los Alamos National Laboratory,
Oak Ridge National Laboratory, Sandia National Laboratories, University of Chicago

Introduction - I/O Forwarding

Modern massively parallel systems exhibit unique I/O architectures and I/O requirements. For example, compute nodes might not have direct outside access or might be running microkernels incapable of fully supporting all I/O functionality. At the same time, contemporary parallel file systems struggle to handle the massive amount of concurrency exposed by these machines. *I/O forwarding* was introduced to tackle these issues.

The basic idea?

> *Instead of performing your own I/O, have it done by some other entity that might be better suited or located.*



Architectural diagram of the 557 TFlop IBM Blue Gene/P system at the Argonne Leadership Computing Facility.

IOFSL

IOFSL – I/O Forwarding Scalability Layer

- Portable I/O Forwarding Implementation
- Production Quality – not just a research project
- Currently provides I/O forwarding on many leadership class machines (IBM BlueGene/P, Cray XT series)

IOFSL provides features not commonly found in other forwarding implementations:

- Flexible extensible design: easy to adapt to new systems and to add support for new filesystems.
- Accelerates I/O research by providing a customizable, open source implementation.
- Manipulation of I/O requests instead of merely forwarding them; used to implement optimizations such as request merging and request scheduling.

MPI Shared File Pointer I/O

> *MPI provides a logical file pointer ("shared file pointer") jointly managed by all the ranks that opened the file.*

The current default implementation in ROMIO uses a file to hold the current value of the shared file pointer, and relies on file locking to ensure non-conflicting access to the file. There are a number of disadvantages to this approach:

- Not all file systems support file locking
- Concurrent write access to the same region of a small file can have a high access cost in a distributed file system.

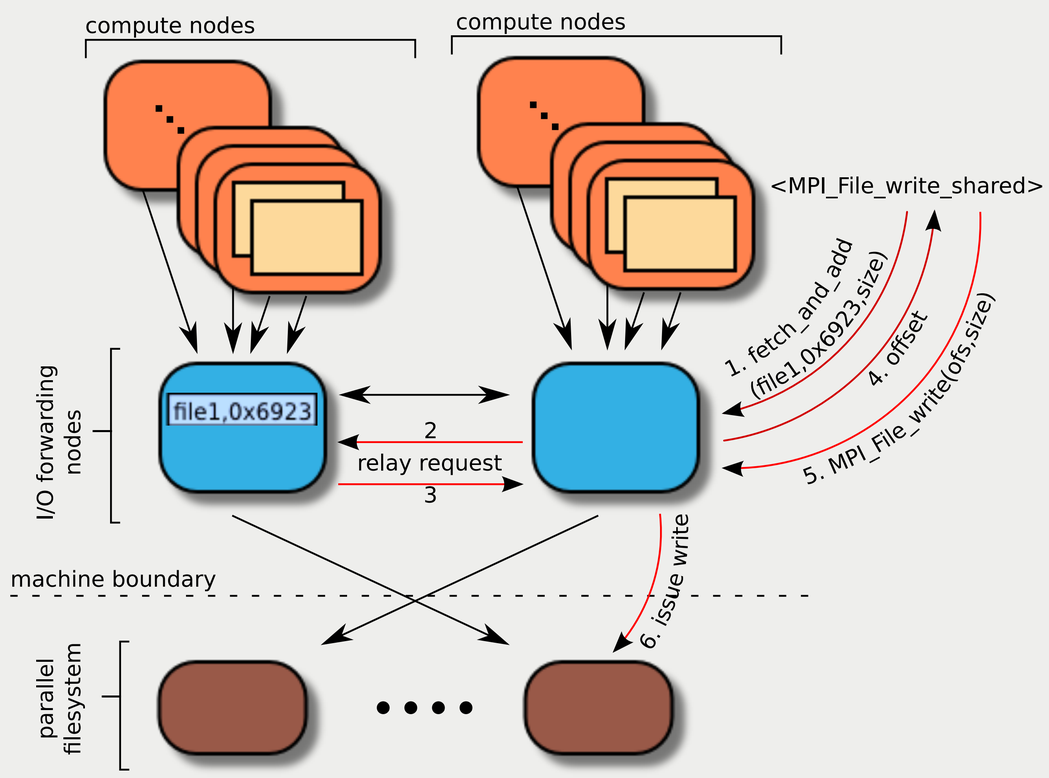
Extended Attributes in IOFSL

We extended `zoidfs_getattr` and `zoidfs_setattr` to support storing and retrieving extra information associated with a file, in a way similar to *extended attributes*.

- These attributes remain in the forwarding server and are never stored on the filesystem (and don't require filesystem support).
- In addition to `CREATE`, `REMOVE`, `GET` and `SET`, an atomic `FETCH_AND_ADD` operation is implemented.

Shared File Pointers using IOFSL Extended Attributes

- At `MPI_File_open` time, a 64 bit cookie is generated and combined with the file handle to form a unique identifier for the shared file pointer. This identifier is broadcasted to all the ranks as part of the open process.
- The shared file pointer (stored in an extended attribute on the forwarding server) is created on demand.
- When the file is closed, rank 0 performs a reduction to query if any of the other ranks created the shared file pointer, and removes it if needed.
- The name space for attributes is shared between the forwarding servers, and a server will relay the request to the "owner" of the attribute when needed. This is important for machines such as BG/P where a compute node can only access the closest I/O node.



Related Work

IOFSL supports a distributed atomic file append mode that is independent of MPI.

- When invoking zoidfs_write, users indicate that the application data should be append to the file.
- When the server receives the atomic append request, it atomically fetches the current end of file position and increments the end of file position. The server stores the application data at the previously identified end of file position.
- The file position is stored in a distributed hash table accessible by all IOFSL servers.
- The server returns the file position that the application data was stored at to the user in a hint.

The Open Trace Format (OTF) toolset uses the IOFSL distributed atomic append capability to store trace data generated by leadership-class applications.

- IOFSL servers aggregate multiple (OTF) trace event streams.
- Multiple OTF trace event streams are are atomically appended into one or more files.
- OTF uses an index of event locations to merge and reorder the files during trace post-processing steps.

Using this file structure and IOFSL's non-blocking I/O features, the OTF and VampirTrace research groups can efficiently generate application traces running at 200,000 cores on the JaguarPF Cray XT5 system at Oak Ridge National Laboratory.

Contributing

We welcome all contributions and collaborations:

- IOFSL Project website: `http://www.iofsl.org/`
- IOFSL Wiki and Developer website: `http://trac.mcs.anl.gov/projects/iofsl/wiki`
- IOFSL Public Git repository: `http://git.mcs.anl.gov/iofsl`

Contact us at `io-fwd-devel@lists.mcs.anl.gov`